The Future Of Package Management

A Cloudsmith Whitepaper

cloudsmith

April 2020

Introduction

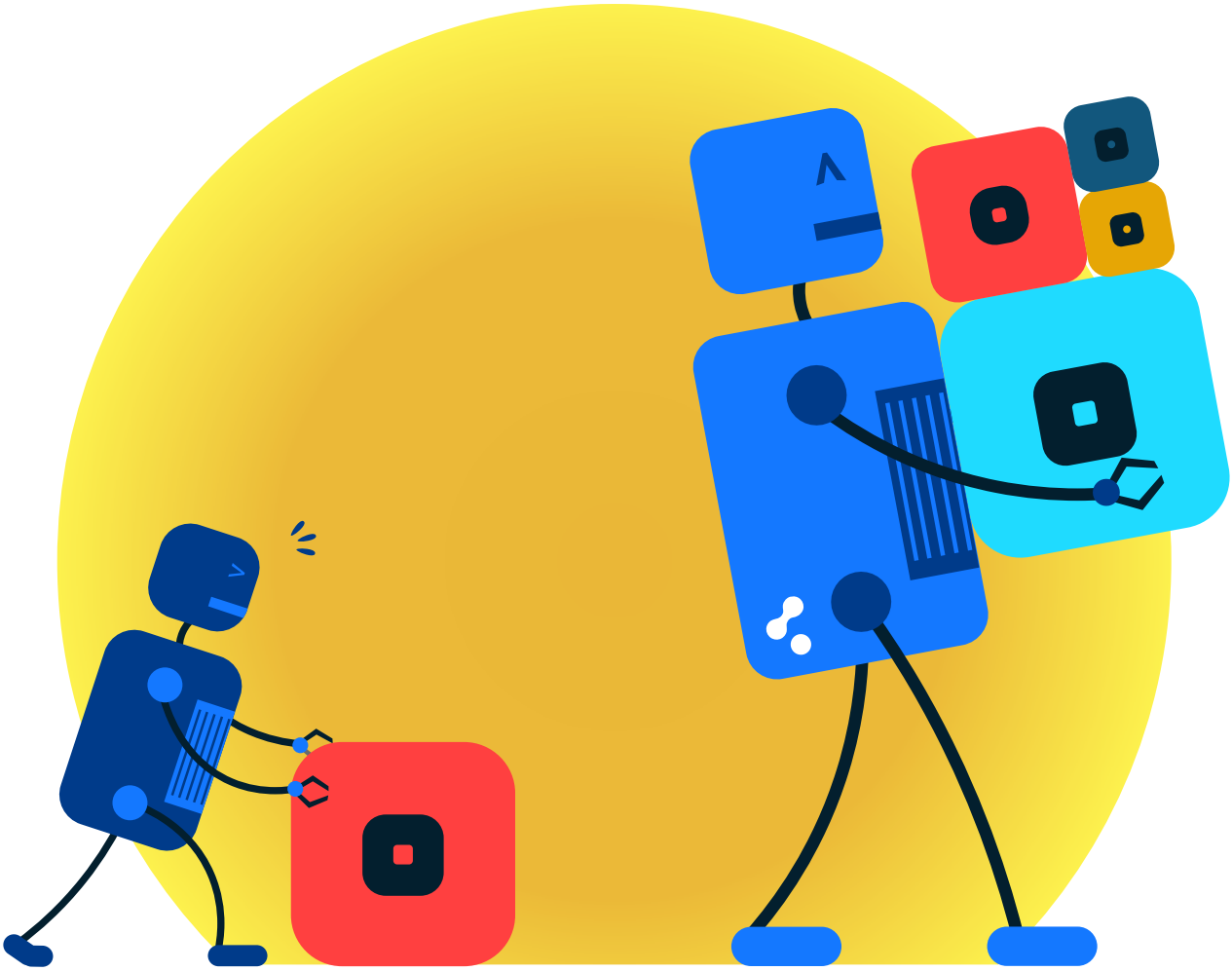
Something is changing in the world of package management.

Most developers and development teams understand the benefits associated with using packages. It beats writing your own code after all. Meanwhile, the cost and risk associated with the management (or lack of management) of those packages were considered to be a price worth paying.

Today, however, it is time to see things differently. New approaches open up the possibility of package management as a strategic benefit to the business: providing a 'central source of truth' for software assets, helping distributed teams work together seamlessly, speeding up development cycles across the board, and even supporting the distribution and sale of commercial software across the globe.

This short paper looks at the challenges organizations face today when it comes to package management, and how this new approach can both address those challenges and create new opportunities.

Enjoy!



Why We Use Packages

> *"If I could wave this magic wand, I would change this step. Instead of getting source code or compiled code from dev through source control,* ***I want packaged code that's ready to be deployed."*** *– William Mason, Director of Quality Assurance, Parts Unlimited, as quoted in The Phoenix Project*

Way back in the early days of software development, it didn't take long before engineers began to realize that they were often writing the same chunks of code over and over again. In the world of source code, a new application might be only a slight variation on one that already existed, but still needed to effectively be written from scratch.

Nobody likes writing something that has been written before. And for this reason, and after some evolution, the idea of the package was born. In this context, a package is an application or library that:

- Performs a specific function
- Can be distributed as a discrete entity, and
- Integrated into new projects in a (relatively) straightforward fashion.

As you can imagine, the existence of packages greatly increases efficiency in delivering software projects, to the extent that an ever decreasing percentage of the average software project today is original code. The rest is pre-existing code, integrated in the form of packages.

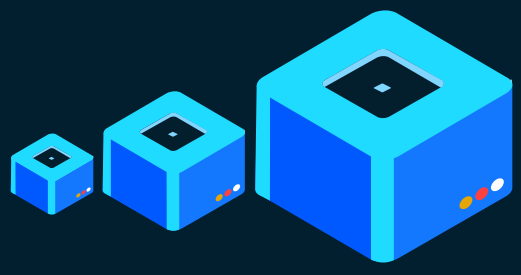
That point brings us to the second key benefit of packaged software, as described so eloquently in chapter 31 of The Phoenix Project and quoted above.

In Eric S. Raymond's seminal essay on open source, The Cathedral and the Bazaar, he defines Linus's Law, which states that "given enough eyeballs, all bugs are shallow." Open-source packages have typically enjoyed the attention of a large number of expert eyeballs. As such they are likely to be bug-free (or more bug-free than anything written in-house) and thus improve overall software quality.

So not only can development teams avoid unnecessary work, they can also have confidence that the packages they use aren't going to break anything (in contrast to original code, which hasn't been tested yet).

For both these reasons, packages are now ubiquitous in the software development process. We use those created by third parties, accessed through public repositories, and we package the code we create ourselves for re-use within our own team.

A Java developer working in Maven, to give one example, can (almost) automatically integrate packages or dependencies from a list of thousands that are available on public repositories. As we've already seen that ability has the potential to greatly speed up the development process. But it comes with risks too.



The Challenges Of Package Management

The mere existence of packages implies we need some way to manage them. In the previous section, we mentioned some of the benefits of packages, but eagle-eyed readers may have spotted the important caveat "if built and distributed correctly". In that short phrase lies an awful lot of detail: detail that is the responsibility of package management.

What does it mean to say that packages are built and distributed correctly? And what happens when they are not?

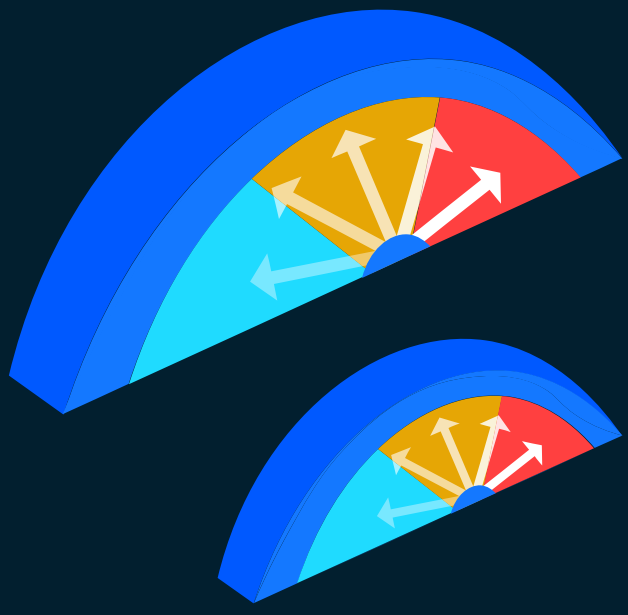
Firstly and most obviously, in extreme, but by no means unprecedented, cases we introduce vulnerabilities into our code that would not otherwise be there. Many packages are accessed from public repositories. Ultimately, we don't write them and we don't control them. So, whilst accepting the truth of Linus's law above, there are always risks involved.

But this issue is in fact just the tip of the iceberg. Ultimately a lack of control leads to a range of issues, each of which can have serious implications for the quality and reliability of the code we develop. Without control we are unable to have a clear insight into which packages the organization is currently using, and where each one is being used. We are unable to control access to packages, or control which versions are in use (and which are not).

Good package management should seek to ensure that we do have control over these aspects of the process. The intention is to move away from the 'Wild West' or free-for-all approach, in which individual developers find what they need and integrate it themselves, and towards a controlled environment in which all the software assets used and needed by the business live.

An environment like this enables us to actively manage our software assets – by integrating them with security scanning for example – and can in turn can guarantee the best possible levels of security, reliability and availability.

> ***"Ultimately a lack of control leads to a range of issues, each of which can have serious implications for the quality and reliability of the code we develop"***



Distribution And DevOps

When it comes to distribution we have a whole other set of challenges. We need to get packages, assets or containers to the people and processes that need them as quickly and reliably as possible.

As modern DevOps processes (specifically continuous integration) might require dozens of builds per day, it is absolutely essential that those builds are quick. In turn, that means whatever packages we use must be integrated into those builds fast.

We have already noted the numbers of packages and dependencies used in modern software projects. When they run into the hundreds within a single build, as they often do, even small increases in latency quickly add up to unacceptable build times. The effect is further compounded by 'chatty' formats (take a bow Maven) that require multiple back-and-forth communications before a package is integrated.

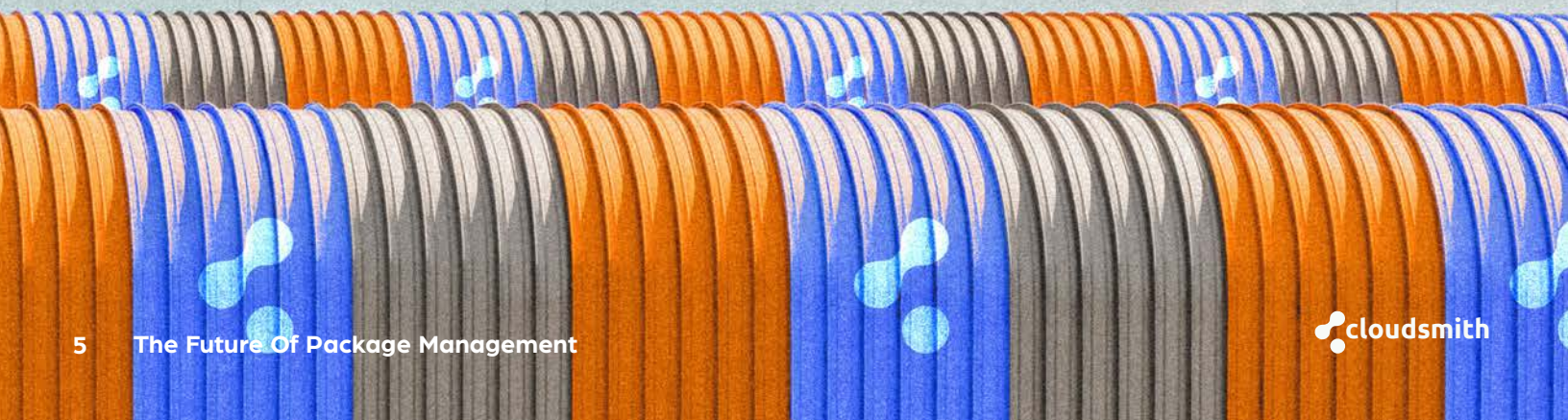
> ***"When dependencies run into the hundreds within a single build, as they often do, even small increases in latency quickly add up to unacceptable build times"***

Particularly when we rely on public repositories, this situation is very difficult to avoid. It takes time to find and download dependencies during development, and performance during build cannot be guaranteed, partly because we may be integrating packages and dependencies from a variety of repositories in a variety of locations. As a result, whilst packages certainly speed up development times, they can be the cause of painfully slow builds.

In response to these challenges, it has become common to bring some form of package management onto the premises – storing packages as close as possible to the development effort. But in a world of distributed teams and multiple locations, where is that effort? Delivering low latency in one location only reduces performance in others.

To date, package management has been an exercise in managing these compromises. The general approach has been one of ameliorating risk and managing cost, rather than one of packaging as strategic value.

We believe that approach has to change.



The Future Vision

What if, as in The Phoenix Project, we waved a magic wand when it came to software packaging? What would we create?

It might look something like this:

- All the packages the organization uses, or may need, are stored in a single location. And in fact I stop thinking of them as 'packages', but rather as assets. The vital building blocks through which all value is ultimately delivered.
- I control that location, so I can verify packages are what they say they are, and ensure they are always available to others
- As a result, I have a single source of truth when it comes to software assets in use in my organization
- Wherever I am in the world, I can integrate those packages into development and deployment processes in a performant fashion
- Whatever I build, wherever I build it, is automatically added to my single source of truth – so it can be re-used in turn.

Let's talk about the characteristics a solution like this would have, and what they mean for the modern software development organization.

> ***"It shouldn't matter what language or format a package was created in, I should be able to store and distribute it through the platform."***

Universal

Firstly and perhaps most obviously, such a system would be universal. It shouldn't matter what language or format a package was created in, I should be able to store and distribute it through the platform. Otherwise, it is hardly a single source of truth.

This is relevant. At present packages tend to be used on an ad hoc basis, often relying on separate solutions for separate languages. That introduces multiple points of potential failure, but more importantly means that at no point do we have a single, clear overview of what is in use and where. Our system changes that. Everything in one place, one integration required, and a single way to handle packages of any description.

To this, one might also add intelligence. Wouldn't it be nice if my universal store of software assets knew what my organization was going to need and ensured it was available ahead of time? After all, if I am storing a package with specific dependencies, then it makes sense to ensure those dependencies are ready to go when the time comes.

And to take things further (we do have a magic wand, remember), we could even use machine learning techniques to anticipate the dependencies and packages that will be required based on user behavior – and again have them ready to roll when the time comes.

Secure

Secondly, this system is secure. In this context, I use secure in the broad sense: by managing all packages within a single, private environment, we have levels of control that in turn greatly reduce the risks associated with using third party assets in our development process.

We can think of this approach as an 'isolation layer'. With all the assets that we use (from whichever source they originally come from, including our own organization) stored in a single environment, we have the control that we talked about above.

This single environment allows us to:

- Quickly and easily **run security scans** on the entirety of the organization's software assets, giving us as much reassurance as is possible (nothing is ever guaranteed) when it comes to vulnerabilities or any form of malware or malicious code getting into our own projects.
- **Control permissions** and be sure that the only people or machines using specific packages are those that should have that ability. Turning that around, a single controlled environment also allows us to shut down the use of packages throughout our entire organization if (when) we become aware of vulnerabilities or other issues with those specific assets.
- **Get a handle on license compliance.** In a free-for-all, in which packages from public resources are integrated into projects on the basis of short-term expediency, the licenses associated with those assets may only get a cursory glance at best. A single environment again gives the organization the space to apply a more methodical approach – and avoid nasty surprises later in the day.
- **Guarantee the availability** of any specific package when we need it. Again, if we store packages in one place, that we control, we know they will be there when required by either a human or a process. As a result, things are less likely to fall over.
- **Measure usage at both the micro and macro level.** One source of truth means we can see who or what downloaded which packages when, whilst also providing account-level analytics relating to the usage of various packages throughout the organization.

> ***"If we store packages in one place, that we control, we know they will be there when required by either a human or a process"***

This is just a short list of examples – there is more. Ultimately they all add up to one thing: the ability of anyone in the team to be able to find and use any software asset with total confidence, with a corresponding dramatic reduction in issues arising from the integration of packages into the software development pipeline.

Cloud-Native

With our last wave of the magic want, let us make our system cloud-native.

We looked earlier at the issues associated with storing software assets on-premises. These can roughly be grouped into two, those relating to performance and those relating to total cost of ownership (TCO).

Performance suffers because in the modern world very few development teams are co-located in a single place. So whilst the lucky few are able to find and integrate packages into builds, the rest are left out in the cold.

Meanwhile, TCO creeps inexorably upward as the implications of being responsible for delivering scalability, availability, reliability and all the rest start to hit home. It should be obvious that if we are building a single source of software assets for an entire business, these attributes are absolutely vital. Guaranteeing such a system stays up and scales to meet demand is emphatically non-trivial, and by extension expensive.

But that's the old world.

The new model of software storage is not just 'in the cloud' but cloud-native, utilizing a content delivery network (CDN) and edge-caching capabilities to respond dynamically to the requirements of the organization and support lightning-fast delivery to any individual, team or process, wherever they are.

> ***"The new model of software storage is not just 'in the cloud' but cloud-native, utilizing a content delivery network (CDN) and edge-caching capabilities to respond dynamically to the requirements of the organization"***

Being cloud-native also enables the entire solution to be run at 'web-scale', delivering the agility and scalability that large organizations need – and which are so clearly not available to anyone managing an on-premises solution.

The benefits of being cloud-native don't stop there. As more and more elements of the software development pipeline move into the cloud, it stands to reason that software assets stored in the same place are easier to integrate into those processes. Cloud-based continuous integration, for example, is going to run faster and smoother with tight integration into a cloud-native package management solution.

A cloud-native approach also helps when it comes to the distribution of commercial assets to customers. After all, if the typical modern development team is distributed across multiple locations, the modern customer base is the same – only more so. And in precisely the same way, a web-scale platform for storing and sharing these assets makes it easier than ever before to get software assets to customers quickly and securely.

Putting It All Together

For too long package management has been characterized by doing 'just enough' to avoid the potentially disastrous consequences of doing nothing.

That is about to change.

If we accept that packages are the building blocks of modern software, the assets that are stitched together to build a new product, then it is time to understand the strategic significance of storing, controlling and distributing those assets in the right way.

When we do so, it becomes possible to greatly improve the efficiency with which we build, deploy and distribute software. We can automate huge chunks of the development process and as a result deliver new levels of reliability and security. Better software, delivered faster, at a lower total cost of ownership.

At Cloudsmith, that is our goal and our vision.

Sign up: cloudsmith.com

Twitter: @cloudsmith

LinkedIn: linkedin.com/company/cloudsmith

Email: support@cloudsmith.io